AMENDMENTS TO THE CLAIMS:

Please amend claims 1-16, 31 and 46 as follows.

This listing of claims will replace all prior versions, and listings, of claims in the application:

1. (currently amended) An apparatus for processing data comprising:

processing logic operable to perform data processing operations; and

an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions; wherein said instruction decoder, in ~~is~~ responsive to a memory access instruction:,

~~(i) to~~ compares a base register value, stored within a base register specified by a base register field of said memory access instruction, with a predetermined null value; and

~~(ii),~~ if said base register value matches said predetermined null value, then said decoder triggers~~to~~ branching to execution of a null value exception handler.

2. (currently amended) An apparatus as claimed in claim 1, wherein in response to said memory access instruction a return address is stored pointing a memory location storing a program instruction to be executed upon a return from said null value exception handler.

3. (currently amended) An apparatus as claimed in claim 1, wherein said null value exception handler is located at a memory address pointed to by a value stored within a programmable configuration register.

4. (currently amended) An apparatus as claimed in claim 3, wherein said programmable configuration register is a coprocessor configuration register.

5. (currently amended) An apparatus as claimed in claim 3, wherein said branch is made to an instruction stored at a memory address given by said value stored within said programmable configuration register subject to a fixed offset.

6. (currently amended) An apparatus as claimed in claim 1, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to emulation of a non-native program instruction that is not directly decodable by said instruction decoder attempting to make a memory access using a null value.

7. (currently amended) An apparatus as claimed in claim 1, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to an error in operation of a virtual machine computer program operable to translate non-native program instructions that are not directly decodable by said instruction decoder into native program instructions that are directly decodable by said instruction decoder.

8. (currently amended) An apparatus as claimed in claim 6, wherein said non-native program instructions are machine independent program instructions.

9. (currently amended) An apparatus as claimed in claim 8, wherein said machine independent program instructions are one of:

Java bytecodes;

MSIL bytecodes;

CIL bytecodes; and

.NET bytecodes.

10. (currently amended) An apparatus as claimed in claim 6, wherein said non-native instructions are native program instructions of a different apparatus for processing data.

11. (currently amended) An apparatus as claimed in claim 10, wherein said processing logic and said instruction decoder are part of a RISC processor and said non-native instructions are native instructions of a CISC processor.

12. (currently amended) An apparatus as claimed in claim 3, wherein said value stored within said programmable configuration register is a start address of said null value exception handler.

13. (currently amended) An apparatus as claimed in claim 3, wherein said value stored within said programmable configuration register is an address of a jump instruction operable to jump execution to a start address of said null value exception handler.

14. (currently amended) An apparatus as claimed in claim 1, wherein said memory access instruction is a load instruction operable to load into a destination register specified by a destination register field within said load instruction a load value dependent upon a value read from a memory location specified by said base register value.

15. (currently amended) An apparatus as claimed in claim 1, wherein said memory access instruction is a store instruction operable to store into a memory location specified by said base register value a store value dependent upon source value stored within a source register specified by a source register field within said store instruction.

16. (currently amended) A method of processing data with an apparatus for processing data having processing logic operable to perform data processing operations and an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions, said method comprising the steps of:

in response to said memory access instruction decoded by said instruction decoder controlling said processing logic:

~~(i) to~~ comparinge a base register value, stored within a base register specified by a base register field of said memory access instruction, with a predetermined null value; and

~~(ii),~~ if said base register value matches said predetermined null value, then triggering~~to~~ branching to execution of a null value exception handler.

17. (original) A method as claimed in claim 16, wherein in response to said memory access instruction a return address is stored pointing a memory location storing a program instruction to be executed upon a return from said null value exception handler.

18. (previously presented) A method as claimed in claim 16, wherein said null value exception handler is located at a memory address pointed to by a value stored within a programmable configuration register.

19. (original) A method as claimed in claim 18, wherein said programmable configuration register is a coprocessor configuration register.

20. (previously presented) A method as claimed in claim 18, wherein said branch is made to an instruction stored at a memory address given by said value stored within said programmable configuration register subject to a fixed offset.

21. (previously presented) A method as claimed in claim 16, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to emulation of a non-native program instruction that is not directly decodable by said instruction decoder attempting to make a memory access using a null value.

22. (previously presented) A method as claimed in claim 16, wherein said null value exception handler is operable to determine if said memory access instruction attempting to

access a location corresponding to a null value corresponds to an error in operation of a virtual machine computer program operable to translate non-native program instructions that are not directly decodable by said instruction decoder into native program instructions that are directly decodable by said instruction decoder.

23. (previously presented) A method as claimed in claim 21, wherein said non-native program instructions are machine independent program instructions.

24. (original) A method as claimed in claim 23, wherein said machine independent program instructions are one of:

Java bytecodes;

MSIL bytecodes;

CIL bytecodes; and

.NET bytecodes.

25. (previously presented) A method as claimed in claim 21, wherein said non-native instructions are native program instructions of a different apparatus for processing data.

26. (original) A method as claimed in claim 25, wherein said processing logic and said instruction decoder are part of a RISC processor and said non-native instructions are native instructions of a CISC processor.

27. (previously presented) A method as claimed in claim 18, wherein said value stored within said programmable configuration register is a start address of said null value exception handler.

28. (previously presented) A method as claimed in claim 18, wherein said value stored within said programmable configuration register is an address of a jump instruction operable to jump execution to a start address of said null value exception handler.

29. (previously presented) A method as claimed in claim 16, wherein said memory access instruction is a load instruction operable to load into a destination register specified by a destination register field within said load instruction a load value dependent upon a value read from a memory location specified by said base register value.

30. (previously presented) A method as claimed in claim 16, wherein said memory access instruction is a store instruction operable to store into a memory location specified by said base register value a store value dependent upon source value stored within a source register specified by a source register field within said store instruction.

31. (currently amended) A computer program product comprising a computer readable storage medium containing computer readable instructions for controlling~~including a computer program operable to control~~ an apparatus for processing data having processing logic operable to perform data processing operations and an instruction decoder operable to decode program

instructions to control said processing logic to perform data processing operations specified by said program instructions, said computer program comprising~~:~~,

in respose to a memory access instruction decodable by said instruction decoder to control said processing logic~~:~~

~~(i),~~ ~~to~~ comparinge a base register value, stored within a base register specified by a base register field of said memory access instruction, with a predetermined null value; and

~~(ii),~~ if said base register value matches said predetermined null value, then ~~to~~ branching to execution of a null value exception handler.

32. (original) A computer program product as claimed in claim 31, wherein in response to said memory access instruction a return address is stored pointing a memory location storing a program instruction to be executed upon a return from said null value exception handler.

33. (previously presented) A computer program product as claimed in claim 31, wherein said null value exception handler is located at a memory address pointed to by a value stored within a programmable configuration register.

34. (original) A computer program product as claimed in claim 33, wherein said programmable configuration register is a coprocessor configuration register.

35. (previously presented) A computer program product as claimed in claim 33, wherein said branch is made to an instruction stored at a memory address given by said value stored within said programmable configuration register subject to a fixed offset.

36. (previously presented) A computer program product as claimed in claim 31, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to emulation of a non-native program instruction that is not directly decodable by said instruction decoder attempting to make a memory access using a null value.

37. (previously presented) A computer program product as claimed in claim 31, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to an error in operation of a virtual machine computer program operable to translate non-native program instructions that are not directly decodable by said instruction decoder into native program instructions that are directly decodable by said instruction decoder.

38. (previously presented) A computer program product as claimed in claim 36, wherein said non-native program instructions are machine independent program instructions.

39. (original) A computer program product as claimed in claim 38, wherein said machine independent program instructions are one of:

Java bytecodes;

MSIL bytecodes;

CIL bytecodes; and

.NET bytecodes.

40. (previously presented) A computer program product as claimed in claim 36, wherein said non-native instructions are native program instructions of a different apparatus for processing data.

41. (original) A computer program product as claimed in claim 40, wherein said processing logic and said instruction decoder are part of a RISC processor and said non-native instructions are native instructions of a CISC processor.

42. (previously presented) A computer program product as claimed in claim 33, wherein said value stored within said programmable configuration register is a start address of said null value exception handler.

43. (previously presented) A computer program product as claimed in claim 33, wherein said value stored within said programmable configuration register is an address of a jump instruction operable to jump execution to a start address of said null value exception handler.

44. (previously presented) A computer program product as claimed in claim 31, wherein said memory access instruction is a load instruction operable to load into a destination register specified by a destination register field within said load instruction a load value dependent upon a value read from a memory location specified by said base register value.

45. (previously presented) A computer program product as claimed in claim 31, wherein said memory access instruction is a store instruction operable to store into a memory location specified by said base register value a store value dependent upon source value stored within a source register specified by a source register field within said store instruction.

46. (currently amended) A computer program product comprising a computer readable storage medium containing computer readable instructions for translating~~including a computer program operable to translate~~ non-native program instructions to form native program instructions directly decodable by an apparatus for processing data having processing logic operable to perform data processing operations and an instruction decoder operable to decode program instructions to control said processing logic to perform data processing operations specified by said program instructions, said native program instructions comprising~~:~~

, in response to a memory access instruction decodable by said instruction decoder to control said processing logic~~:~~

~~(i) to~~ comparing~~ed~~ a base register value, stored within a base register specified by a base register field of said memory access instruction, with a predetermined null value; and

~~(ii)~~, if said base register value matches said predetermined null value, then triggering~~to~~ branching to execution of a null value exception handler.

47. (original) A computer program product as claimed in claim 46, wherein in response to said memory access instruction a return address is stored pointing a memory location storing a program instruction to be executed upon a return from said null value exception handler.

48. (previously presented) A computer program product as claimed in claim 46, wherein said null value exception handler is located at a memory address pointed to by a value stored within a programmable configuration register.

49. (original) A computer program product as claimed in claim 48, wherein said programmable configuration register is a coprocessor configuration register.

50. (previously presented) A computer program product as claimed in claim 48, wherein said branch is made to an instruction stored at a memory address given by said value stored within said programmable configuration register subject to a fixed offset.

51. (previously presented) A computer program product as claimed in claim 46, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to emulation of a non-native program instruction that is not directly decodable by said instruction decoder attempting to make a memory access using a null value.

52. (previously presented) A computer program product as claimed in claim 46, wherein said null value exception handler is operable to determine if said memory access instruction attempting to access a location corresponding to a null value corresponds to an error in operation of a virtual machine computer program operable to translate non-native program instructions that are not directly decodable by said instruction decoder into native program instructions that are directly decodable by said instruction decoder.

53. (previously presented) A computer program product as claimed in claim 51, wherein said non-native program instructions are machine independent program instructions.

54. (original) A computer program product as claimed in claim 53, wherein said machine independent program instructions are one of:

Java bytecodes;

MSIL bytecodes;

CIL bytecodes; and

.NET bytecodes.

55. (previously presented) A computer program product as claimed in claim 51, wherein said non-native instructions are native program instructions of a different apparatus for processing data.

56. (original) A computer program product as claimed in claim 55, wherein said processing logic and said instruction decoder are part of a RISC processor and said non-native instructions are native instructions of a CISC processor.

57. (previously presented) A computer program product as claimed in claim 48, wherein said value stored within said programmable configuration register is a start address of said null value exception handler.

58. (previously presented) A computer program product as claimed in claim 48, wherein said value stored within said programmable configuration register is an address of a jump instruction operable to jump execution to a start address of said null value exception handler.

59. (previously presented) A computer program product as claimed in claim 46, wherein said memory access instruction is a load instruction operable to load into a destination register specified by a destination register field within said load instruction a load value dependent upon a value read from a memory location specified by said base register value.

60. (previously presented) A computer program product as claimed in claim 46, wherein said memory access instruction is a store instruction operable to store into a memory location specified by said base register value a store value dependent upon source value stored within a source register specified by a source register field within said store instruction.

61-63. (cancelled)